Quality improvement of requirements specification via automatically created object oriented models

Daniel Popescu
Georgia Institute of Technology

Abstract

In industry most software requirements specifications are written in natural language. Software analysts prefer natural language over formal languages because it increases the communication between all stakeholders. However, the downside is that natural language specifications are often imprecise and ambiguous. This paper explores whether natural language processing can help to reduce ambiguity and to increase precision in natural language specifications. To examine the possibilities of this approach, a CASE tool was implemented. The developed approach uses syntactical knowledge to build an object oriented analysis model. Since the automatically extracted diagram is smaller than the original specification and clearly visualized, it can enhance cognition. Therefore, a human can more easily find ambiguities tracing the defects back to the original specification. A case study illustrates and evaluates the developed approach.

1 Introduction

Daniel Berry suggests three steps to improve the quality of natural language specifications. [3]

1. Learn to write less ambiguously and less imprecisely.
2. Learn to detect ambiguity and imprecision.
3. Use a restricted natural language which is inherently unambiguous and more precise.

The described work addresses the second point.

In industry most software analysts write requirements specifications in natural language. This approach enhances the communication between all the stakeholders, because everyone can understand clearly written natural language text. However, the downside is that natural language specifications are often imprecise and ambiguous.

An object oriented analysis model is one way of verifying a natural language specification. During software development a software analyst creates this model, before he designs the software architecture. However, often a different team creates the software design, therefore the requirements specification should already be precise and clear, when the software designer receives the specification document.

This work tries to develop an approach to detect ambiguities in natural language specifications. In detail, it tries to answer the following research questions. Can an object oriented analysis model, which is created automatically, help to increase the quality of requirements specifications? Can automation help detecting ambiguities? Since the creation of an object oriented analysis model is part of most object oriented development methodologies, we believe that it is a suitable verification model for requirements specifications.

A software tool can detect ambiguities in a requirements specification because of its strengths compared to the limitations of a human. A software tool can scan, search, browse, and tag huge text documents faster than a human analyst. Furthermore, a tool works rigorously. In contrast, a human analyst can easily overlook mistakes, since he assumes that they are correct. Berry calls this *subconscious disambiguation*. The reader understands an interpretation and think that it is the only one.

The second section describes the developed approach. Many object oriented methodologies [15] suggest a syntactical text analysis to extract the object oriented analysis model. Classes are created from nouns of the text, verbs are transformed into associations or class methods. Since human analysts use mainly syntactical heuristics to create the analysis model, a software tool can create one with the help of a syntactical analysis.

Syntactical analysis and natural language parsing is a complex topic on its own. However, Berry's above mentioned third suggestion addresses this problem. This work assumes a controlled grammar for the natural language requirements specification.

A controlled grammar enforces constrains on a natural language. It only allows a subset of the grammar rules of English. Therefore, it increases structure without seriously compromising readability. A computer can more easily apply rules on controlled grammars and any non-technical stakeholder can still understand this restricted natural language. Therefore, the requirements specification does not lose expressiveness. This approach uses a controlled grammar, which is based on a controlled grammar developed by Moreno et al. [10]. Their grammar is used to transfer requirements specification into object-oriented domain models. Additionally, they also suggest transformation rules. In their work a human analyst performs the transformation from text to an object oriented analysis model. This work extended the Moreno's et al. grammar to allow a less constrained grammar which allows more English grammar constructs.

Section three presents a case study of the developed approach. It discusses the creation of a model from a natural language specification that described an elevator scenario.

Section four deals with domain specific terminology in depth. For extracting class candidates, technical terminology is one of the most important challenges that this software tool has to overcome. Most domains have their own terms, abbreviations and vocabulary. Therefore, a tool must detect these terms to create the correct classes. It must be capable of producing classes like *elevator button* or *DMP organizer*. The classes *button* or *organizer* would be insufficient for the object oriented analysis model.

Additionally, in section four a simple approach to detect technical terminology is presented. The metrics precision and recall evaluate the performance of the technical term extraction, which is performed on an example technical document. For an automated tool a high recall value is more important than a high precision value, since the output is controlled by a human analyst afterwards. A human usually has a high precision in his work and can filter noisy data. In contrast, if a tool does not detect some classes, a human analyst would have to redo the work manually to detect this error. Since a human analysts has worse recall, the error might stay undetected.

Before the related work is discussed, we want to answer one more question. Why should a software analysts use this tool only for verification and not for producing an object oriented analysis model? This approach is based on natural language processing and not on natural language understanding. A tool cannot judge whether the produced model describes a good set of requirements or classes. This requires understanding. Berry argues tools that run by themselves take the human out of the loop and make it less likely that a human will think about the results.

1.1 Related Work

The discussed related work can be divided into four partitions. First, work is introduced about natural language processing applications on requirements specifications. Second, a discussion of controlled grammar follows. Third, methodologies and tools are introduced about automatically extracting an object oriented analysis model. Fourth, a work about domain specific terminology is presented.

Kof describes a case study on the application of natural language processing to extract terms, to classify them and to build a domain ontology [11]. This work is the most similar to our approach. The domain ontology, which can be created with Kof's method, consists of nouns and verbs, which constitute the concepts within the domain ontology. In the end, the domain ontology helps to detect weaknesses in the requirements specification. Vincenzo Gervasi and Bashar Nuseibeh report on their experiences of using lightweight formal methods for the partial validation of natural language requirements documents [6]. They check properties of models obtained by shallow parsing of natural language requirements. Furthermore, they demonstrate scalability of their approach with a case study of a NASA specification.

Fuchs and Schwitter developed Attempto Controlled English (ACE) [4]. ACE is a subset of English that can be unambiguously translated into first order logic. Over the last years, this controlled grammar evolved into a mature controlled grammar, which is mainly used for reasoning about requirements specifications [5]. Moreno et al. developed another controlled grammar [10]. They defined a formal correspondence between linguistic patterns and conceptual patterns. Following this approach, an engineer has to write a specification according to a controlled grammar (called SUL and DUL). Afterwards he can easily create an object oriented models, using the mapping rules that this paper supplies.

Several implementations exists of tools that automatically transform a requirements specification into an object oriented analysis model. Mich developed the CASE tool NL-OOPS[12]. In her approach, before creating the object oriented model, the parsed language is transformed into a semantic network. Afterwards, the object model is derived from the semantic network. Mich's CASE tool is the only long-term research project in the area of automatically creating object oriented analysis models. Barker and Biskri present an implementation of an approach that uses only syntactical extraction rules [2]. Harmain and Gaizauskas present also an implementation of such a CASE tool [8]. Furthermore, they introduce a method to evaluate the performance of such CASE tools.

Mollá et al. developed a method for answering questions in technical domains. This work recognizes the importance of technical terminology for natural language processing tools [13].

2 Approach

The goal of our approach is to improve the quality of natural language specifications. Since this work uses object oriented analysis model to find ambiguities in the specification, a way is needed to extract this model from the natural language specification.

This approach exploits syntactical information for building classes, associations and attributes. Common heuristics suggest that we build classes from nouns, methods from verbs, and attributes from adjectives. Abbott described the relation between linguistic and formal representations in 1983 [1]. Since this relation can be transformed into a powerful and easy-to-understand design strategy, object oriented methodologies adopted this approach [15].

However, natural languages are rich in ways to express the same concepts differently. The same meaning can be expressed with various grammar constructs. For example, a sentence in active voice can be translated into passive voice without changing the semantics or pragmatics. However, passive voice can encourage ambiguities. For example, the sentence "The illumination of the button is activated" leaves room for different interpretations, because it is not clear who holds the responsibility of activating the button. Furthermore, it could describe a state. As a consequence, controlled grammars can be introduced to decrease the possibility of ambiguities. They enable formal reasoning without having the disadvantages of new formal languages [5]. Controlled grammars provide the other advantage that they can be better parsed, and extraction rules can be created more easily.

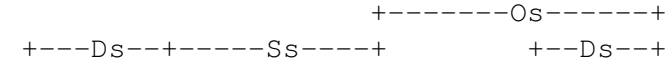
This work uses a controlled grammar that is derived from Moreno's et al. grammar [10]. In their work they have developed two free context grammars called the Static Utility Language Grammar and Dynamic Utility Language Grammar. Based on these grammars they developed an unambiguous mapping to object oriented models. This mapping is explicitly defined and allows better model creation than common heuristics that are more justified intuitively. Furthermore, the explicit definition enables automation.

Controlled grammars enforce simple sentences. Most sentences have a basic structure consisting of subject, verb and object. Furthermore, only simple sub clause constructions are allowed like conditional clauses ("If ...", "When ..."). Therefore, a requirements specification contains many short sentences, if it was written accordingly to the controlled grammar.

The parser we used was developed by Daniel D. Sleator and Davy Temperley at Carnegie-Mellon University [16]. Richard Sutcliffe and Annette McElligott showed in their work that the link grammar parser is a robust and accurate parser for software manuals [17]. Since software manuals are similar to software specifications, it promises a high parsing accuracy for our work. The used parser is based on the theory of link grammars. A link grammar consists of a set of words (the terminal symbols of the grammar), each of which has one or more linking requirements. A sequence of words is a sentence of the language defined by the grammar if there exists a way to assign links among the words so as to satisfy the following three conditions:

1. Planarity: The links do not cross;
2. Connectivity: The links suffice to connect all the words of the sequence together;
3. Satisfaction: The links satisfy the linking requirement of each word in the sequence.

The link grammar parser produces the links for every sentence. After parsing, the links can be accessed through the API of the link grammar parser. In our approach the links are used to create rules. The following example shows three different link types. The input sentence is "The elevator illuminates the button."



```
|       |           |        |      |
the elevator.n illuminates.v the button.n .
```

A **D** link connects a determiner to a noun, an **S** link connects subject nouns to finite verbs and an **O** link connects transitive verbs to their objects. The small **s** after each connector shows that the link connects to a singular noun.

From this example sentence, the first extraction rule can be derived. If a sentence contains an **S** link and an **O** link then create a class from the subject noun and one from the object noun. Afterwards, create a directed association from the subject to the object class, which is named after the verb. A directed association was chosen over a simple method, because an association shows who invokes the action. If the action would have been modeled as a method, the information would have been lost who causes the action. A directed association was chosen over an undirected to avoid misinterpretation of the reading direction.

Using this rule, the tool would extract the classes *elevator* and *button* with a directed association *illuminate* from the *elevator* class to the *button* class.

To avoid having two classes created for *elevator* and *elevators*, our approach incorporates *WordNet* to find the stems of nouns and verbs. WordNet is an online lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory.

Finally, the extracted object oriented analysis model is transformed into a visual representation. For this step, we use the tool UMLGraph. This tool declaratively describes UML models. It produces a *dot file*, which the tool *Graphviz* can transform into popular graphic formats like JPEG, GIF or PNG. Figure 1 shows an overview over the automated part of our approach.

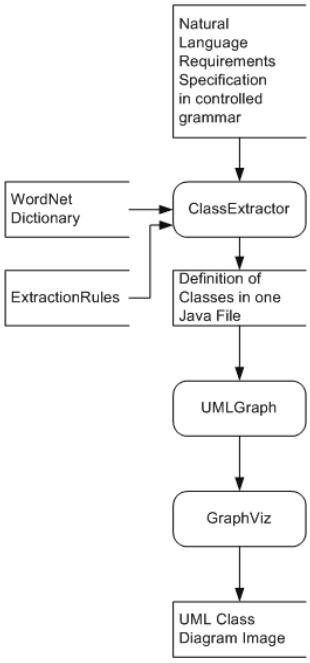


Figure 1. The implementation of the developed approach.

2.1 Rules

The used extraction rules were derived from Moreno's et al. grammar [10], object oriented methodologies literature [15] [14], and conducted experiments.

In total the system uses 13 extraction rules. The five most frequently used are the following:

1. The most frequent applicable rule is the above described rule. If a parsed sentence contains a subject and an object link, then create two classes with an directed association named after the verb.

2. Aggregations are an important concept of UML class diagrams. One rule for extracting aggregations is similar to the first rule. The major difference is the verb. This rule is only applicable, if a parsed sentence contains a subject and an object link and the verb stem is either *have*, *possess*, *contain*, or *include*. In that case, the object is aggregated to the subject.

3. Sometimes subclauses describe actions without the need of an object. This can happen if the system reacts to a given event. "If the user presses the button, the elevator moves." An event clause starts with an *if* or a *when*. If the system detects an event clause and the main clause has only a subject link, then a class from the subject link noun is created and the verb is added to the new class as a method.

4. Genitive attributes indicate two classes with an aggregation. "The system stores the name of the customer." or "The system stores the customer's name.". If this system detects a genitive, it creates two classes with one linking aggregation. In this example, it would create a class *customer* and it would aggregate the class *name* to the class *customer*. *Name* could have been modeled as an attribute. However, other sentences in the specification could add methods to the class *name* later. Therefore, using only syntactical information of one sentence it is undecidable if *name* is an attribute or an aggregated class at this point.

   This rule needs to be constrained by semantic information. For example, the system should not apply this rule to the sentence "The user enters the amount of money." Although *amount* is a noun, a class *amount* is not desired in this case. Another rule extracts the desired classes *user*, *money* and the association *enter* from this sentence.

5. Although active clauses are preferred in natural language specifications, passive clauses are still needed. They are used to describe relations and states. For example, a "A husband is married to his wife.".

   From the first sentence two classes are created from the subject noun and the noun of the prepositional phrase. The passive verb and the connecting word *to* to the prepositional phrase describe together the association.

Two post processing rules are applied after all possible extraction rules are executed. One rule converts aggregations into attributes and the other removes the class *system* from the diagram.

The first rule converts classes that are aggregated to another class in attributes of that other class. Only classes that lack any attributes, methods, or further incoming and outgoing associations are transformed. For example, one rule described above extracted two classes and one aggregation from the sentence "The system stores the name of the customer.". The rule creates a class *name* and a class *customer*. However,

the class *name* has probably no method or association. Therefore, if it contains no method after all rules were applied, it is transformed into an attribute of the class *customer*.

The second post processing rule removes the class *system* from the analysis model, since it describes all other classes as a superset and is therefore no entity on its own.

2.2 Interpretation

After the object oriented analysis model is created, a human analyst can check it for ambiguities. The following list give some ideas that a human analyst can use.

- The associations are a hint for possible ambiguities. For example, if two different class send a message to the same class. It should be guaranteed that they actually communicate with the same class. If a motion sensor activates one type of display and a smoke detector actives another type of display, then the class diagram should reflect that with two different display classes.
- Each class should reflect one concept. For example, it should be checked if *book* and *textbook* are really two different classes, when the system did not create a generalization between these two classes.
- If a class has an attribute, but the attribute is not a primitive type like a string or a number, a definition of the attribute might be missing in the original text. After the definition is added, the attribute should be reflected by an own class.
- If a class has no association, it could indicate that no relations or interactions between this class and other classes were specified. This could be an underspecification.

3 Case Study

An elevator specification is used as a case study to demonstrate our approach [9].

However, before the tool could create the object oriented analysis model, the specification had to be transformed to meet the controlled grammar constrains.

The original specification looked like the following: An n elevator system is to be installed in a building with m floors. The elevators and the control mechanism are supplied by a manufacturer The internal mechanisms of these are assumed (given) in this problem.

Design the logic to move elevators between floors in the building according to the following rules:

1. Each elevator has a set of buttons, one button for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited (i.e., stopped at) by the elevator.
2. Each floor has two buttons (except ground and top), one to request an up-elevator and one to request a down-elevator. These buttons illuminate when pressed. The buttons are canceled when an elevator visits the floor and is either traveling the desired direction, or visiting a floor with no requests outstanding. In the latter case, if both floor request buttons are illuminated, only one should be canceled. The algorithm used to decide which to serve first should minimize the waiting time for both requests.
3. When an elevator has no requests to service, it should remain at its final destination with its doors closed and await further requests (or model a ”holding” floor).

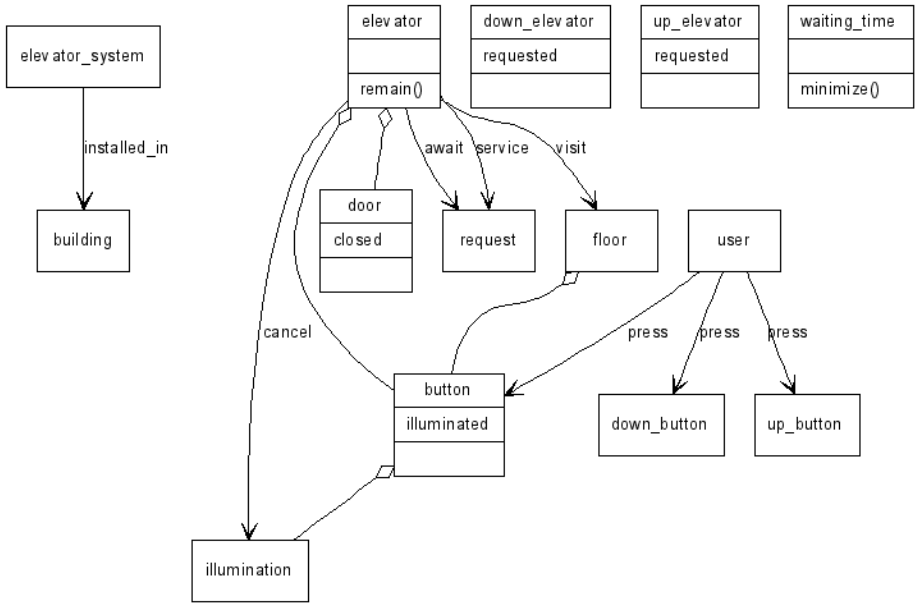


Figure 2. The automatically created analysis model from the elevator example.

4. All requests for elevators from floors must be serviced eventually, with all floors given equal priority.

5. All requests for floors within elevators must be serviced eventually, with floors being serviced sequentially in the direction of travel.

After the specification is transformed into the controlled grammar, it looks like the following.
An elevator system is to be installed in a building with floors.

1. Each elevator has buttons. Each elevator has one button for each floor. When a user presses a button, the button becomes illuminated and the elevator visits the corresponding floor. When the elevator visits a floor, the elevator cancels the corresponding illumination.

2. Each floor has two buttons. (except ground and top). If the user presses the up-button, an up-elevator is requested. If the user presses the down-button, a down-elevator is requested. If the user presses a button, this button becomes illuminated. When the elevator visits a floor, the elevator cancels the corresponding illumination of the button in the desired direction. The system minimizes the waiting time.

3. When an elevator has not to service any requests, the elevator remains at its final destination and the doors of the elevator are closed. The elevator awaits further requests.

4. The elevators service all requests from floors eventually with equal priority.

5. If a user presses a button within the elevator, the elevator services this request eventually in the direction of travel.

The implemented tool creates the diagram Figure 2 out of the second specification.

After applying the above mentioned guidelines, the diagram reveals four defects in the natural language specification.

1. The diagram shows a class for an *up-elevator* and a *down-elevator*. Both classes have no connection to any other class, nor do they have any association to the class *elevator*. Furthermore, the *up-elevator* has an attribute *requested*, while the *elevator serves a request*, which indicates that the *up-elevator* and *down-elevator* are a specialization of the *elevator*. All this indicates that the concepts *up-elevator* and *down-elevator* are not defined enough.

2. The class *doors* in the diagram contains the attribute *closed*. However, the class has no methods to manipulate this state. If closing and opening the door is within the scope of the system, it indicates that the concept *door* is not defined enough.

3. Both, the floor and the elevator, have buttons. Therefore, the class *elevator* and the class *floor* have an aggregated class *button*. However, the diagram indicates that they have the same type of button. Since they have different behavior, it is unlikely that the same class describes both. Talking only about *button* could lead to different interpretations. Therefore, defining a concept *elevator button* and *floor button* would enhance lucidity in the specification.

4. The class *up-button* and the class *down-button* are only connected to the class *user*. Since the *user* is an actor in the system, the diagram does not clarify where the buttons belong to. It can be derived from the natural language specification, because they are mentioned in the same paragraph as the floor. However, it is not necessary to assume this. Therefore, both concepts can be clarified to reduce the ambiguities in the specification.

This case study points out that the automated approach has to recognize special domain specific terminology. Only identifying the class *button* would not be sufficient. Special classes are needed for *up-button*, *down-button*, *elevator button* and *floor button*. A tool should also not ignore concepts that contain abbreviations like *DMP7 elevator button*. Therefore, it is important that the tool reaches a high recall value for domain specific terms.

4 Domain specific terminology

The developed approach uses only nouns as a source for classes. Since a valid object oriented analysis model is needed as sub goal for a quality improvement of the natural specifications, a high noun recall value is required.

To evaluate the performance of the term extraction, the metrics *recall* and *precision* are measured for a technical manual. These metrics are used for a variety of natural language processing systems [7] and are used as an indicator for the success of natural language information extracting systems. The metrics are defined as follows.

- **Recall** shows how well a system can identify pieces of information compared to the total amount of pieces of information in the source. It is defined as follows.

$$Recall = \frac{I_{correct}}{I_{total}}$$

Here $I_{correct}$ defines the correct identified pieces of information and I_{total} defines all correct pieces of information in the source text.

- **Precision** shows how accurate a system identifies pieces of information.

$$Precision = \frac{I_{correct}}{I_{correct} + I_{incorrect}}$$

  Here $I_{correct}$ defines the correct identified pieces of information and $I_{incorrect}$ defines identified pieces of information that are not correct.

For this approach a high recall value is more important than a high precision value, since human analysts can identify incorrect data. However, if information is missing, it is hard to determine if the tool did not recognize the data or if the data is not present in the document.

As mentioned above for the identification of classes a high recall value for nouns in a text is desired. However, the noun on its own is not sufficient. If a tool extracts only the concept *button* from *elevator button* it is an incorrect identification. A tool must decide if a noun is the base for a term or only an attribute for another noun. For identification of compound nouns, we can use the **AN** and **G** link of the link grammar parser. Since most domain specific terminology is build up from attributive nouns (AN) or proper nouns (G), it already improves the recall value of the terminology detection.

In general, parsers have problems parsing words that are not in their internal dictionary. The link grammar parser has a guessing mode, where it can guess the syntactical function of a unknown word. Therefore, it can also guess some domain specific terminology.

The following experiment shows how well technical terms are identified with the help of the link grammar parser.

The *intro man page* of the *Cygwin environment* is used as a source for the experiment. The steps of the experiments are as follows.

1. We manually extracted all terms, a noun or a compound noun, from the *man page*. It contained 52 terms like *Cygwin*, *Linux-like environment*, *Linux API emulation layer* and *POSIX/SUSv2* among others. These 52 terms consists of 33 single noun terms and 19 compound noun terms.

2. For the first experiment the link grammar parser extracted every term out of the *intro man page* without the capability of extracting compound noun terms. It recognized 31 of the single noun terms and none of the compound noun terms. Therefore, it reached a recall value of 59.6%.

3. For the second experiment the compound noun term detection was added to the link grammar parser. After this, the tool recognized 10 compound noun terms. As the single noun terms detection rate stayed the same, the tool recognized 41 terms. Therefore, it reached a recall value of 78.8%.

Afterwards the undetected terms were examined. It turned out that five terms were undetected because the appeared in grammatical obscure or wrong parts in the sentence. Correcting these sentence, increased the detected terms to 46 and the recall value to 88.46%. The tool generated seven wrong terms, since it created wrong terminology for the incompletly detected terms. Therefore, it reached a presion value of 86.79%.

The five not identified terms were *case-insensitive file system*, *intro man page*, *Linux look and feel*, *Red Hat, Inc.* and *User's Guide*. *Red Hat, Inc.* is not recognized because of the comma, *User's Guide* cannot be detected syntactically, because if every genitive was part of a term, it would lead to an overgeneration of terms. A frequency analysis might help in larger documents to improve the term recall.

Linux look and feel is not recognized because of the *and*. *case-insensitive file system* and *intro man page* can only be partial detected, because *case-insensitive* is an adjective, which is only sometimes cases part of a term. Another example demonstrate the difficulties caused by adjectives. In *readable manual* only *manual* is a term in most cases. The term *intro man page* is not recognized, because the link grammar parser guesses that *intro* is an adjective. However, if it is planned that *case-insensitive file system* is a concept within the requirements specification, then writing it upper-case would allow the link grammar parser to detect it. An upper-case word in the middle of a sentence indicates that it is a proper noun and therefore the link grammar parser can identify it as a term.

This example shows that domain specific terminology does not need to be a problem for our approach.

5 Conclusion

This work demonstrated that an automatically created object oriented analysis model can help to reduce ambiguities in a natural requirements specification. A case study confirmed our hypothesis. Critical factors for the success of this approach are a controlled grammar and the capability of identifying domain specific terminology. The controlled grammar allows it to produces rules and to improve the parsing performance. Without the capability of identifying the approach would never be applicable for industry projects.

While working on our implementation and the terminology experiment, the tool outperformed the recall value of the investigators working manually, because they had overlooked something. A carefully conducted review session could also find the overlooked classes and terms. However, running a tool is certainly cheaper and faster than a formal review session.

Still, an automated approach cannot replace a manually crafted analysis model. The syntactical information are a good source for the model creation, but semantic knowledge is required. For example, a *request* could be a class or a method. This depends not only on the domain, but also on the architectural decision for the model. In general, it is questionable if it is desired to take the human out of the loop [3]
.

Although the software industry has not accepted or developed solutions for automated quality measurement for natural language specification, we believe that this approach can already be applied. More studies have to be conducted to confirm our hypothesis. Furthermore, this approach would only be one part of a natural language processing framework, since a natural language specification has more sources of ambiguities than this work introduced.

5.1 Future Work

CASE tools for natural language processing of specifications are an interesting, important and challenging approach. Therefore, many possibilities for future research exist.

- How does the tool perform on larger (industrial) natural language specifications? Those results would help to explore the problem space and to find new unsolved research questions.
- Our implementation cannot resolve anaphora. Anaphora is the use of a linguistic unit, such as a pronoun, to refer back to another unit. ”The customer can buy text books and return *them*”. *Them* is an example for an anaphora, which the automated tool must resolve.

- Domain-specific terminology identification can be improved. As mentioned above, syntactical information is not sufficient to detect all the terms within a document. Therefore, frequency analysis might improve the performance.
- Further semantic knowledge could improve the capability of a tool. For example, the lexicon WordNet contains information about hypernyms which indicate superclasses and meronyms which indicate aggregations. This information could be used to show missing links in an object oriented analysis model. However, although WordNet is a large online lexicon, it lacks domain specific terminology and therefore might only be a little help. Extending the dictionary with domain-specific terminology could reduce this problem.
- The current implementation is not applicable on natural requirement specifications that have a functional language style ("The system must provide the functionality of ..."). This would require another controlled grammar. Future work could examine which grammar is the most suitable for class extraction.
- The UML offers a set of different diagram types. Natural language processing could be used to create sequence, state or other diagrams. For example, Moreno et al. [10] developed also a controlled grammar for specifying dynamic behavior.
- Other work has found different sources of ambiguities in natural language specifications. Since there seems not to be a single approach, different sets can be combined into a framework in validating natural language specifications. This could lead to a new methodology of developing requirement specifications.

References

[1] Russel J. Abbott. Program design by informal English descriptions. *Communication of the ACM*, 26(11):882–894, 1983.

[2] D. Barker and K. Biskri. Object-oriented analysis: Getting help from robust computational linguistic tools.

[3] Daniel Berry. Natural language and requirements engineering - nu? In *www.ifi.unizh.ch /groups/req/IWRE/papers&presentations/Berry.pdf*, accessed on 2.12.2005.

[4] N. Fuchs and R. Schwitter. Attempto controlled english (ACE). In *The First International Workshop On Controlled Language Applications*, 1996.

[5] Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto controlled english (ACE) language manual, version 3.0. Technical Report 99.03, Department of Computer Science, University of Zurich, August 1999.

[6] Vincenzo Gervasi and Bashar Nuseibeh. Lightweight validation of natural language requirements. *Softw., Pract. Exper.*, 32(2):113–133, 2002.

[7] Ralph Grishman. Information extraction: Techniques and challenges. In *SCIE '97: International Summer School on Information Extraction*, pages 10–27, London, UK, 1997. Springer-Verlag.

[8] H. M. Harmain and Robert J. Gaizauskas. CM-builder: A natural language-based CASE tool for object-oriented analysis. *Autom. Softw. Eng.*, 10(2):157–181, 2003.

[9] Mats Heimdahl. An example: The lift (elevator) problem. http://www-users.cs.umn.edu/ heimdahl/formal-models/elevator.htm, accessed on 14.12.2005.

[10] Natalia Juristo, Ana Maria Moreno, and Marta Lpez. How to use linguistic instruments for object-oriented analysis. *IEEE Softw.*, 17(3):80–89, 2000.

[11] Leonid Kof. Natural Language Procesing for Requirements Engineering: Applicability to Large Requirements Documents. In Alessandra Russo, Artur Garcez, and Tim Menzies, editors, *Automated Software Engineering, Proceedings of the Workshops*, Linz, Austria, September 21 2004.

[12] Luisa Mich and Roberto Garigliano. NL-OOPS: A requirements analysis tool based on natural language processing. In *Proceedings of Third International Conference on Data Mining Methods and Databases for Engineering*, Bologna, Italy, 2002.

[13] Diego Moll, Rolf Schwitter, Fabio Rinaldi, James Dowdall, and Michael Hess. NLP for answer extraction in technical domains. In *11th EACL 2003, Proceedings of the Conference*, 2003.

[14] Sastry Nanduri and Spencer Rugaber. Requirements validation via automated natural language parsing. *Journal of Management Information Systems*, 12(3):9–19, Winter 1995-96.

[15] J. Rumbaugh. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[16] D. D. Sleator and D. Temperley. Parsing English with a link grammar. In *Third International Workshop on Parsing Technologies*, 1993.

[17] Richard Sutcliffe and Annette McElligott. Using the link parser of Sleator and Temperley to analyse a software manual corpus. In *Industrial Parsing of Software Manuals (pp. 89-102)*, 1995.